Disk Fragmentation Issues Explained

Craig Jensen, CEO
Vaughn McMillan, Research Director
Executive Software International

1 June 1993

Disk Fragmentation Issues Explained

1 Overview

This report discusses several different, yet related issues. It is intended to provide information that will help you make informed decisions about key aspects of your VAX™ or Alpha AXP™ computer system.

First, this report will examine the most widespread system performance problem of all — disk fragmentation. Uncorrected, disk fragmentation will eventually slow your VAX or Alpha AXP computer to intolerable levels. This report will briefly explain the cause of and the various cures for disk fragmentation.

Next, we will explore the practicality of file optimization — that is, the placement of files in specific locations on a disk to improve system performance.

Finally, we will discuss steps you can take to avoid some of the effects of disk fragmentation by taking preventative measures on your system.

2 Fragmentation — The Biggest Performance Problem

Before we go too far, a definition of *fragmentation* is in order.

The word fragmentation means "the state of being fragmented." The word *fragment*, means a "detached, isolated or incomplete part." It is derived from the Latin *fragmentum*, which in turn is derived from *frangere*, meaning "break." So, *fragmentation* means that something is broken into parts that are detached, isolated or incomplete.

There are two types of fragmentation with which we are concerned: file fragmentation and free space fragmentation. *File fragmentation* refers to files stored on a computer disk that are not whole but rather are broken into scattered parts, located in separate areas on the disk. *Free space fragmentation* means that the empty space on a disk is broken into scattered parts rather than being collected into one big empty space.

File fragmentation causes problems accessing data stored in computer disk files, while free space fragmentation causes problems creating new data files or extending old ones.

Taken together, we refer to the two types of fragmentation as *disk fragmentation*.

By its very nature, the OpenVMS™ operating system allows (and in fact causes) fragmentation to occur. On today's VAX and Alpha AXP systems, this has become a problem.

The original VAX/VMS™ operating system (and in fact the current OpenVMS for both VAXes and Alpha AXPs) was deliberately designed to allow fragmentation and to deal with it efficiently. This design was actually the solution to an older and more serious problem.

2.1 The Original Problem

When computers were first introduced, they had no disks at all. Obviously, disk fragmentation was not a problem back in those days. Nobody had even conceived of the idea yet, so the operating systems in use at the time had no mechanism to deal with fragmentation at all.

These historic days of computer use extended into the late 1960's, before the VAX and even before the PDP-11 on which the VAX was based. In the Digital world, these were the days of the PDP-8.

Eventually, disks came along. The first disks were small — incredibly small by today's standards. They were measured in thousands of bytes rather than millions, not to mention the billion-plus byte capacities we see today.

Nevertheless, the early disks were seen as a fabulous advance in the days when memory sold for a dollar a *byte*. (Right now, if you shop around you can find memory for about $50.00 per megabyte — that is 20,000 times less than the old days.)

Later, the early PDP-11 operating system, called RT-11 (for Real Time-11), was capable of storing data on disks in files, and the files were organized in a formal file structure. This file structure, however, *required* that all files be contiguous — no file could be split into two or more pieces. It was not a question of performance; the system simply did not have the capability to create or read a file that was split into pieces. A file had a single location on the disk, and that was that.

This requirement for contiguous files meant that a newly-created file had to fit within a single gap of free space on the disk, or it would not be created at all. It was not possible to allocate a part of the file to one gap and the rest of the file to another, as can be done with today's OpenVMS systems. This was true even when the total amount of free space on the disk far exceeded the size of the new file. There had to be a large enough *contiguous* free space, period.

As you might assume, this problem of not enough contiguous free space to create a file occurred every time a disk filled up, and small disks fill up *very* fast. When files were deleted frequently, it was not unusual to have a disk reach a point where no more files could be created, yet the disk was only a little more than half full.

2.2 Fragmentation . . . The Solution?

Before long, a great new operating system came along, one that allowed multiple simultaneous users of the same PDP-11 computer — RSX-11 (which stood for Resource Sharing eXecutive).

The designers of RSX-11 created a clever file system that had the revolutionary (at *that* time) capability to locate parts of a file in different places on the disk. Each file had a *header*. The header gave the location and size of each piece of the file, so the file could be in pieces scattered around the disk. With this breakthrough, a file could be created anytime there was sufficient free space *anywhere* on the disk; the free space did not have to be contiguous.

Back in those days, there was no drawback to this mechanism whatsoever. It really *was* "a feature, not a bug." Performance losses due to fragmentation, even when taken to extremes, caused very little difficulty for anyone. Keep in mind though, at this time in the early 1970's disks were very small. Now look at a real example:

The RK05 disk, which was in common use at the time, held 2 1/2 megabytes (5,000 blocks) of data. Suppose the disk was completely and utterly fragmented, in other words, no two consecutive blocks of data were contiguous. Every single disk access likely required moving the head and waiting for rotational latency. (*Rotational latency* is the delay caused by the situation where the read/write head is not directly over the needed area on the disk, and therefore must wait for a revolution of the disk to complete before performing the disk operation.) Even under these conditions, the entire disk could be read in 250 seconds (50 milliseconds average access time times 5,000 blocks). That is a little over four minutes of time, even in a worst case scenario.

The same action on a modern 700 megabyte disk, even with a 16 millisecond access time, takes over 6 *hours*.

Keep in mind that over the same period of time, disk capacities have increased 500 times that of the RK05 and CPU speeds have increased 400 times that of the original PDP-11. Unfortunately, even though disk speeds have increased by nearly a factor of three (from 50 milliseconds to about 15 to 20 milliseconds average access time) they have not kept pace with the rest of computer technology. This makes the speed of the disk a major bottleneck in the computer system.

Clearly, today's larger disks brought with them more than faster speeds and larger capacities. They brought a susceptibility to a new kind of computer disease — fragmentation.

The ability to deal with fragmented files, which was carried over from RSX-11 to the OpenVMS operating system, was a solution to an earlier problem that failed to anticipate the enormous capacities to which disks would grow. There is no end in sight. Deliberate fragmentation is no longer a feature — it *is* a bug.

3 How Fragmentation Occurs

In order to understand how OpenVMS causes files to become fragmented, it is important to be aware of the mechanism which lets the system know where the free spaces on the disk are located. In quick, general terms, there are three mechanisms used by the OpenVMS file system to find free space. These are the *extent cache*, the *BITMAP.SYS* file, and the *bitmap* cache.

3.1 The Extent Cache

The *extent cache* is the key factor causing file fragmentation on OpenVMS systems. It is, in simple terms, a list in system memory (on each node of the cluster) of disk blocks available for file allocations. This list is made up of the spaces where recently-deleted files previously resided. The extent cache contains a list of known free *extents* on the disk that can be allocated when a file is newly created or extended. An extent is simply a pointer consisting of a Logical Block Number (LBN) and a size, that points to a logically contiguous area of disk space. The extent cache is the first place checked by the operating system when looking for a place on the disk to create or extend a file. By default, this cache contains 64 extents, but this number can be changed. Also, this number will vary when the total free space on the disk is 10% or less of the total volume size.

3.2 The BITMAP.SYS File

The BITMAP.SYS file (or storage bitmap file) is the operating system's map of all the disk's blocks. In essence, each bit in the file represents minimally a block (but usually more) of disk space. With this map, the system can see which areas on the disk are allocated and which are free. With larger disk capacities, it is inefficient for the computer system to deal with millions of individual blocks one at a time. In this situation, BITMAP.SYS is too large to be useful unless single bits in the map can represent more than one disk block. For this reason, disk blocks are grouped into *clusters*, which are then read and written to as a single unit. On an OpenVMS system, a cluster is the minimum amount of disk space that can be allocated for data storage. The cluster size, in terms of the number of blocks that make up a cluster, is established when the disk is initialized.

3.3 The Bitmap Cache

The *bitmap cache* (or storage bitmap cache) contains a listing of blocks from the storage bitmaps of volumes mounted on the system. The bitmap cache resides in system memory, making it almost instantly accessible. Each bit in the bitmap represents one disk cluster for a given volume. The bitmap cache is used by the Extended QIO Processor (XQP) to avoid the need to read the BITMAP.SYS file from disk when a file extent is requested.

3.4 What Happens?

Whenever a file is deleted, the locations and sizes of the various pieces of the deleted file are placed into the extent cache.

Later, when a file is created or extended, the operating system first scans the extent cache to fulfill the allocation request. If need be, the new file (or file extension) will be broken into pieces to fit in the locations pointed to by the extent cache.

In the event that the extent cache does not point to enough locations to fulfill the allocation request, the operating system will then scan the bitmap cache.

If the bitmap cache scan *still* does not contain enough disk space to fulfill the allocation request, the entire BITMAP.SYS file is scanned.

Due to the very nature of an active, multi-user system, the extent cache usually points to variably-sized free spaces that are scattered randomly around the disk. If the most-recently deleted file on the disk was *larger* than the new file (or file extension), the new file (or the file extension) will be stored contiguously on the disk. But, if the most-recently deleted file on the disk was *smaller* than the new file or file extension, the new file will be immediately fragmented.

When you consider the long-term effects of this type of allocation strategy on a disk in continuous use, you can readily see that fragmentation becomes extreme.

It gets worse — fragmentation at its worst comes in the form of a multi-header file. As the name implies, this is a file with more than one file header. To be more precise, it is a file with a header containing so many retrieval pointers they will not fit into a single one-block header. (Retrieval pointers are the part of the header that shows the logical block locations of the various fragments.) OpenVMS, therefore, allocates a second (or third or fourth) block in the INDEXF.SYS file to accomodate storage of the extra retrieval pointers. In case you are curious, the first file header holds somewhere around 70 pointers (depending on the size

of the file fragments they are pointing to), and the subsequent headers hold around 102 pointers each. Thus, by the time a third file header is created, you have somewhere in the neighborhood of 172 fragments for a *single file*. This borders on the ridiculous.

4 How Widespread Is It?

How widespread is this fragmentation thing? Doctors use the word *pandemic* to describe a disease when virtually everyone has it. Fragmentation is without a doubt pandemic on OpenVMS systems. The only exceptions to this would be:

- a diskless VAX or Alpha AXP system
- a VAX or Alpha AXP with preformatted, read-only disks (like a CD-ROM)
- a VAX or Alpha AXP that is turned off

On a system that is not used very often, fragmentation will be slight — plus, if you do not use it, who cares about fragmentation?

This leaves us with all the other OpenVMS systems out there — the vast majority by far. These systems are typically running 24 hours a day, used interactively by the users from around 8:00 AM to the vicinity of 6:00 PM, typically with usage peaks around 10:00 AM and 2:30 PM, and a dead spot around lunchtime. Most of these systems have sporadic activity during the early evening, then slam into 100% utilization around midnight when a pile of batch jobs start up and run for several hours. Then usage tapers off to nearly nothing until early morning, when the users arrive, starting the whole cycle again.

This type of system typically has several disk drives dedicated to user applications. These disks usually get a lot of use, with hundreds of files being created and deleted every day. Naturally, more files are created than deleted, so the disk tends to fill up and stay that way, until the system manager forces the users to delete excess files.

Under circumstances like these, a disk will fragment badly. Typically you will see a 10% to 20% increase in fragmentation each week. In other words, if you had 10,000 files and all of them were contiguous at the beginning of the week, in a one-week period you could expect those 10,000 files to consist of 11,000 pieces or more. A week later, you would be up to 12,000 or more. After a month, fragmentation would exceed 40% with over 14,000 pieces. In three months, fragmentation is over 240%, with over 34,000 pieces. After a year, those same 10,000 files could theoretically be fragmented into somewhere around 1.4 million pieces. It could not get this bad, since there are not enough disk blocks to hold that many pieces (at least not on this "typical disk"). Besides, the performance would get so bad that users would not be able to use the system, you would get fired, and you would not care how fragmented the disk is anymore.

It *is* true, however, that a poorly managed disk, with nothing done to handle the fragmentation, will become so badly fragmented over time that it becomes for all practical purposes unusable. This is because each file is in so many pieces that the time necessary to retrieve all the files a user needs is just not worth the effort.

5 How Do You Know If You Have Got It?

Now that the disease has been described, you may be wondering to yourself "Do *I* have fragmentation? How do I know if I've got it?"

Some system managers just know they have it, while others wait for the users to complain loudly about the rotten performance of the system, and use *that* as a measure of fragmentation. The real professional system managers actually investigate *before* it becomes a problem.

It is not difficult to find out if you have fragmentation or not, although it *is* easier if you have the right tools.

5.1 VMS Tools to Use

There are several VMS utilities you can use to get a limited picture of the fragmentation on your disks. The information they provide is somewhat scant, but they are free and you have them already.

5.1.1 The DUMP Utility

The simplest and most direct way to see if a file is fragmented is to use the DUMP command:

```
$ DUMP /HEADER /BLOCKS=END:0 file_specification
```

This command allows you to examine the header of a specific file to see if it contains multiple retrieval pointers. (This information is at the bottom of the dump output.) Each retrieval pointer represents a piece of a file. If you only see one pointer, the file is contiguous; if you see two pointers, the file is fragmented into two pieces, and so on.

The limitation with this command is that it can only be used with one file at a time. It would not be prudent for you to go through all the files on a typical disk in this manner.

5.1.2 The MONITOR Utility

One way to determine whether fragmented files are causing your system to do excessive disk I/O is to use the MONITOR IO command:

```
$ MONITOR IO
```

This command will cause a display to be shown giving a variety of I/O system statistics. Of particular interest is the fourth line, the *Split Transfer Rate*. This line tells you how many times the system is having to do two or more I/O transfers when one would serve. A split transfer is the result of fragmentation. If you see any number higher than 0, you are suffering performance losses due to fragmentation.

Another similar MONITOR command is MONITOR FCP. (*FCP* stands for *File Control Primitive*).

```
$ MONITOR FCP
```

The critical line in this display is the *Window Turn Rate*, which tells you how many times OpenVMS had to load new retrieval pointers from the file's header to gain access to the desired portion of the file. The term *window* in this case means the set of retrieval pointers the system keeps in memory to access the file. If the file is contiguous, only one pointer is needed to access the file, so a window turn would never occur. A window typically holds seven pointers, so a file can be fragmented in seven pieces and still be accessed without a

window turn. When there are eight or more fragments, however, one or more pointers have to be flushed and new pointers are loaded to get at the later parts of the file. If a file is fragmented into many pieces, window turns can become a major performance bottleneck.

Split transfers and window turns are not the only consequences of fragmentation, but they are the only ones you can detect with the MONITOR utility that comes with the OpenVMS operating system.

5.2 Other Digital Utilities

Digital has other utilities that have some fragmentation analysis capability, such as DECperformance Solution (or DECps), which is actually a combination of components of the now-discontinued Software Performance Monitor (SPM) and VAX Performance Advisor (VPA). Although discontinued, SPM and VPA are still in use at some sites. Any of these three utilities will tell you a limited amount of information about the fragmentation on your system, but they will not tell you much, since they are intended to tell you a lot of other performance-related information. In addition, they are fairly expensive, particularly compared to some of the fragmentation analysis utilities available from third party vendors.

5.3 Third-Party Utilities

Several third-party software companies (including Executive Software) sell or give away free utilities that perform an analysis of your disks, then provide reports showing a variety of information. Among this information is:

- Free Space Information, including:
  - » Total Free Space Size
  - » Number of Free Spaces
  - » Smallest Free Space
  - » Largest Free Space
  - » Average Size of Free Space
- File Information, including:
  - » Maximum Number of Files
  - » Total Number of Files
  - » Total Size of all Files
  - » Smallest File Size
  - » Largest File Size
  - » Average Size of all Files
  - » Number of Extension Headers
  - » Total File Fragments
  - » Average Fragments per File
  - » Number of Files with 2 or More Fragments
- Lists of the Most Fragmented Files

- Summarized Information about the Free Space Distribution

and a number of other details about the files and free spaces on the disk.

A utility providing this information can be acquired at no cost from Executive Software. It is known as the Disk Analysis Utility.

One of the key pieces of information the Disk Analysis Utility provides is a figure representing the percentage (or degree) of fragmentation on the disk. With the Disk Analysis Utility, this figure is presented as a decimal number. For example, a figure of 1.2 means that 20% of the files on the disk are in two or more pieces, 1.3 means 30%, and so on. This figure is referred to as the *Fragmentation Index*.

6 Why Is It Bad?

With all this discussion about what fragmentation is and how to find it, sooner or later the question comes up: "So what's the big problem? So my files are fragmented . . . it can't be all *that* bad."

Well, that isn't so. If fragmentation is left unhandled, your VAX or Alpha AXP could become useless. If the Disk Analysis Utility indicates that 20% or more of your files are fragmented (this is a badly fragmented disk), you may be in trouble. You had better do something about it *fast*, before the system stops altogether.

In the case of a fragmentation index showing 20% of your files are fragmented, this indicates that your computer has to do perhaps 20% more work to process those files. It should be pointed out that these numbers are merely *indicators*. If only a few files are badly fragmented while the rest are contiguous, and those few files are never accessed, the fragmentation may have no performance impact on your system at all. On the other side of the coin, if those few badly fragmented files are the ones your users or applications are accessing constantly, the performance impact could be much more than 20%. You have to look further to be sure. For example, if there were 1,000 files on the disk and only one of those files is ever used, but that one is fragmented into 200 pieces (20% of the total fragments on the disk), you have a serious problem much worse than the 20% figure would indicate. In other words, it is not the fact that a fragmented file causes performance problems, it is the computer's attempts to access the file that degrade performance.

To fully understand this, it is first necessary to examine how files are accessed and what is going on inside the computer when files are fragmented.

6.1 What Happens to Your Disks

Without going into too many details about disk operation, allow me to explain the fragmentation problem from the viewpoint of the disk.

As you are probably aware, disks platters are divided into concentric circles called tracks. Data is stored within the confines of these tracks. The disk spins at high speed, and the read/write head (which is mounted on an arm much like the tone arm on a phonograph) moves from track to track as needed as the disk spins.

If a file is contiguous, the contents of the file are likely to be within a single track. The data in the file can be scanned from the disk in one continuous sweep merely by positioning the head over the correct track and then detecting the file data as the platter spins the track past the head.

Now picture a file broken into two pieces, but both pieces are still within a single track. To access the file, the head has to move into position, scan the first part of the file, then suspend scanning briefly while waiting for the second part of the file to move under the head. Then the head is reactivated and the rest of the file is scanned.

As you can imagine, the time needed to read the fragmented file is longer than the time needed to read the contiguous (or unfragmented) file. The exact time needed is the time to rotate the entire file under the head, *plus* the time needed to rotate the gap under the head. A gap like this might add a few milliseconds to the time needed to access a file. Multiple gaps would, of course, multiply the time added. The gap portion of the rotation is purely wasted time. Then on top of that, you have to add all the extra operating system overhead required to process the extra I/Os generated by the multiple reads.

Next, look at a file that is fragmented into two parts, but this time the two parts are on separate tracks. Now, in addition to the delay added by the rotation of the disk past the gap, we have to add time for the movement of the head from one track to another. This track-to-track movement is usually much more time-consuming than rotational delay, costing tens of milliseconds per movement. Further, this type of fragmentation is much more common than the gap form (where the fragments are on the same track).

To make matters worse, the relatively long time it takes to move the head from the fragment in the first track to the fragment in the second track *can* cause the head to miss the beginning of the second fragment, necessitating a delay for nearly one full rotation of the disk, waiting for the second fragment to come around again to be read.

Now here is the frightening thought — files do not always fragment into just two pieces. You might have three or four, or a dozen, or even a hundred fragments in a single file. Imagine how hard your disk heads are working trying to collect up the pieces of a file that is in a hundred fragments!

6.2 What Happens to Your Computer

The previous section discussed what happens to your disks as the result of fragmentation; now examine the other system overhead caused by fragmentation.

As mentioned earlier, when a file is in two or more pieces, a separate I/O operation is needed to retrieve each fragment. These are called *split transfers*.

If the file is fragmented into more than the seven pieces that can be accommodated by a single file window, and the eighth (or later) fragment is accessed, one or more retrieval pointers are flushed from the window and reloaded with more retrieval pointers. That is called a *window turn*.

And finally, when more than 70 or so pointers are required in the file header to map a file, a second (or third, or fourth, etc.) file header is required, making this a *multi-header file*.

Each of these fragmentation symptoms costs system overhead, and each one described costs more than the one before.

For every split transfer, you add the overhead of a second (or third, or fourth, etc.) disk I/O transfer. For every window turn, you add the overhead of reloading the window, in addition to the I/Os required just to access the fragments. For every multi-header file accessed, you add to each I/O the overhead of reading a second (or third, or fourth, etc.) file header from the INDEXF.SYS file.

On top of all that, I/O requests due to split transfers and window turns are added to the I/O request queue along with the ordinary and needful I/O requests. The more I/O requests there are in the I/O request queue, the longer users and applications have to wait for their I/Os to be processed. This means that fragmentation causes *everyone* to wait longer for I/O, *not* just the user accessing the fragmented file.

As you can see, fragmentation overhead definitely mounts up in a big way. Just picture what it is like when there are 300 users on the system, all incurring similar amounts of excess system overhead!

7 What Can Be Done?

This leads us to the obvious question — "What do I do about fragmentation?" Get rid of it, of course.

How? There are a number of ways, ranging from some basic system management tactics that do not fully get rid of it but help, to fully automated, complete solutions to the problem. The bottom line is that something *can* be done about fragmentation.

7.1 Clear Off Disks

First, you could keep your disks half empty. If you could enforce this as policy, it would keep enough free space so files would not fragment so badly on creation. It is the moment of file creation that the fragmentation problem begins. When the disk is nearly full, the free space on the disk tends to fragment badly. This greatly increases the likelihood that a newly created file will be created in many small fragments. When the disk is about half empty, the free space tends to occur in larger pools, increasing the chances that the new files will be created in a contiguous piece, or at worst, in only a few larger fragments.

Of course, this so-called solution has the drawback of having to have twice as much disk space as you really need. Nice if you have the budget, and hard to enforce anyway, since nature abhors a vacuum — space on a disk will inevitably get filled.

7.2 COPY /CONTIGUOUS

The second solution is to use the DUMP /HEADER command to examine files that are known to be in heavy use. Then, when a heavily fragmented file is found, use the COPY /CONTIGUOUS command to defragment the file, purging the old copy once the new one is made. This is a simple and inexpensive solution, but very tedious to say the least. It has the added drawback of changing the creation and backup dates of each file copied, which means that your incremental backups are going to swell with files that have not materially changed. It also changes the File ID for the file, which may cause problems for your batch and print queues. Further, you must be very sure not to attempt this operation at the same time an application is accessing the file. At best, the application could be locked out of the file and may abort processing. At worst, the application could update the *old version* of the file during your copy-making and the updates would be lost in the purge. Another major problem is alias file names. OpenVMS allows a file to have two or more names, called *aliases*. The file must not be open when the COPY /CONTIGUOUS command is used, but it is possible, if the file has an alias, that the file could be open under the alias and so this technique could fail.

7.3 Backup and Restore

From the time Digital's users first started complaining about fragmentation, the officially recommended remedy was an operation called "backup and restore". The "and restore" part of this operation is the catch. Omitted from the "backup and restore" phrase is the critical middle step — *initialize*.

"Backup" is easy. You do that anyway (or at least you *should* be doing that). So this step does not take any more time than you already spend just to make sure your files are safely backed up. Backing up a disk does nothing for fragmentation. You have to reinitialize the disk and restore all the files to the disk. *That* is the part that fixes the fragmentation.

Initializing the disk, of course, effectively deletes all the files from the disk. The data can then be restored from the backup tape (or other media) and the data is restored to a clean, unfragmented state.

There are drawbacks to this solution, too. Not the least of these is the time it takes to restore the files to the disk. This takes just about as long as the backup process itself, which is not exactly quick.

Another drawback is the absolute requirement that the backup be precisely accurate in every respect. If the tape is badly flawed or if the drive malfunctioned, your data is lost. You simply cannot get your data back. Therefore, you are strongly encouraged to verify your backup tapes before initializing the disk. The verification pass, of course, takes quite a long time, too.

The most aggravating aspect of backup and restore is the fact that it must be done outside of prime system time. You cannot very easily erase all the users' files when they are using them, and they tend to get upset when you deny them access to their files during the workday. So you come in at night or over the weekend to do your backup and restore. And this goes on for *hours*, even for a few disks.

Then there is the often forgotten aspect — the cost of performing backup and restore. If you are taking disks off-line during production, you are losing the income-producing work that would have been done by that disk. If you are waiting until after hours to take the disk off line, you (or an operator) are working additional hours, and that has got to cost a certain amount. Plus, during the time you are doing backup and restore it is hard to get any other useful work done since you keep having to jump up and change tapes. When you look at all the aspects, it can be quite costly.

7.4 Defragmentation Software Products

Nowadays, there are software products available that you can use to defragment disks. These are referred to as *defragmenters*. Defragmenters come in two forms: off-line defragmenters and on-line defragmenters.

7.4.1 Off-line Defragmenters

The original defragmenters were designed as off-line defragmenters, meaning that you had to take the disk out of service (off-line) to use the defragmenter on it.

Why? This type of defragmenter analyzes the disk to determine the state of fragmentation and then maps out a rearrangement of the files on the disk that will reduce or eliminate the fragmentation. After mapping out where the files should go, it rearranges them. This type of defragmentation has to be done off-line to accommodate the drawbacks inherent in this type of method.

The biggest problem has to do with the separate analysis pass followed by the file rearrangement pass. If, after calculating the ideal position for each file on the disk, some user application were to come along and delete a file, add a new file or extend an existing file, the analysis would be instantly obsolete and the planned rearrangement would be unlikely to produce ideal results. In fact, with some of the early defragmenters, rearranging files with an obsolete analysis was (and still is) downright dangerous. If the defragmenter were to write data into an area it thinks is free but has become occupied since the analysis, user data could be lost. By taking the disk out of service, so no user application can access the disk, this danger is eliminated.

Fortunately, because a better solution came along, off-line defragmenters have all but disappeared from the market, although there are still some in use today.

7.4.2 On-line Defragmenters

Currently, most if not all of the defragmenters on the market are designed to run on a disk without the need for taking the disk out of service. There are essentially two defragmentation strategies: Those that perform an analysis pass over the entire disk, then a defragmentation pass to defragment the files, and those that look at each file individually, determines where to move it, does that, then goes on to the next file.

The on-line defragmenters that use the analysis pass followed by the file movement pass are a holdover from the off-line defragmenter days. The developers of these defragmenters have devised mechanisms to alleviate the dangers of file activity while the defragmenter is running. Unfortunately, if a file is changed between the analysis pass and the file movement pass, that file will not be defragmented. This type of defragmentation strategy is no longer an issue of danger, it is an issue of efficiency and effectiveness.

With some of these defragmenters, the analysis pass can take quite a long time, especially on a high capacity disk. The chance for a file to change between the analysis pass and the defragmentation pass increases with the length of time required to perform the analysis pass. The problem with the increased time is not that the defragmentation is not done quickly, it is that the longer it takes, the less effective it is.

Some of the defragmenters that use this strategy consume a sizable amount of system resources while performing the analysis pass and the file movement pass. If there are users (or batch processes) active on the system while the defragmenter is running, the utility that is supposed to be improving system performance is actually impeding the system, at least while the defragmenter running. A defragmenter is of no use if the system has to work harder defragmenting files that it would to simply retrieve the fragmented files. This would be a case of the cure being worse than the disease!

The other approach to on-line defragmentation is for the defragmenter to analyze each file individually, and then handle that particular file. This is the approach taken by Executive Software with their DISKEEPER/*Plus*® customized online disk defragmenter.

Like the defragmenters that analyze the entire disk then move the files, DISKEEPER/*Plus* will refrain from moving any file if a user application tries to open the file. The difference here is that DISKEEPER/*Plus* typically performs several passes through all the files on the disk, one at a time, before waiting until the next scheduled run time. Chances are, if DISKEEPER/*Plus* backs off a file because a user application opens the file, the file will be handled on the next pass.

Defragmenters using the "one file at a time" strategy (such as DISKEEPER/*Plus*) typically use fewer system resources to do their job, since they are not amassing information about the entire disk in a single scan. This means that DISKEEPER/*Plus* can run while other processes are active on the system, without getting in their way. Obviously, *some* resources are used, but typically not enough to slow down the users.

Some of the manufacturers of the "analysis pass/defragmentation pass" defragmenters point out that their product defragments the disk more quickly than the "one file at a time defragmenters". In some cases, this may be true, but with defragmentation, speed is not necessarily a critical issue. Speed *is* important if your defragmenter brings the system to its knees while it is running, but speed does not matter much at all if the defragmentation process does not slow down the users.

Some system managers schedule their defragmenter to run at night, so it does not get in the way of the users. Resource consumption becomes less of an issue, unless there are batch jobs in progress. Batch jobs are just as important as users, except they do not call you up and complain that the system is slow.

Here again, if the defragmenter does not impede the users, it is not likely to get in the way of batch processes. The bottom line is resource consumption, day *or* night.

All of the on-line defragmenters end up with similar end results — the disk is defragmented. That is where the similarity ends. There are the two main strategies to on-line defragmentation, as have been pointed out. In addition, all of the defragmenters offer a variety of features, utilities, interfaces, and so on. Each vendor will try to convince you that theirs is the only one worth considering.

Since they all end up doing essentially the same thing, judge the product on things like ease-of-use, documentation, additional utilities, and in general, the look-and-feel of the product. Do not forget to consider technical support — this gets overlooked too often. Consider the complete picture, and you will see why DISKEEPER/*Plus* is the best-selling third party software for VAX computers of all time, and the *only* defragmenter (at this writing) for Digital's Alpha AXP systems.

8 What About Optimization?

Then there is the "optimization" theory. For years, the issue of "disk optimization" has been a hotly debated subject within the OpenVMS community. Once the facts are examined though, it becomes clear that "optimization" is not quite optimum.

Those promoting the optimization theory maintain that placing frequently accessed "hot" files at some "optimum" location on a disk can save system resources and time by reducing the distance the disk read head must travel when retrieving these "hot" files from the disk.

The opposing argument follows the idea that in a real-world environment, with users and applications regularly creating, accessing, expanding, and deleting files, the so-called "optimum" scenario is constantly changing. Additionally, keeping up with and reacting to these changes requires more system resources than the original "optimization" scheme was designed to save.

Some of the defragmenters on the market for VAX computers attempt to optimize files. But defragmentation and optimization are two separate issues.

Disk defragmentation alone improves performance of any OpenVMS system. Fragmented files are the number one reason for slower and slower response time. As already pointed out, if the files on a disk are contiguous, access to those files is quicker. This is due to the reduction in disk I/O operations necessary to retrieve the file. When a file is fragmented, one disk I/O is needed to retrieve each piece of the file. If a file is fragmented into five pieces, five disk I/Os are necessary to retrieve the file; ten fragments mean ten disk I/Os. A file broken up into only *two* fragments *still* takes 100% more time to retrieve from the disk than a defragmented file. Since disk I/O is the most time-consuming action involved in file retrieval, reducing the number of disk I/Os substantially reduces the time necessary to retrieve a file. When you consider that each fragment can take 15 to 20 milliseconds (or more) to retrieve, each fragment you eliminate saves you substantial time. In this way, disk defragmentation speeds up system performance.

Optimization, on the other hand, attempts to take defragmentation one step further by placing frequently accessed files in a position on the disk where they can be accessed quickly by the disk read head. By carefully placing files near the center of the disk (away from the outside or inside edges), the access time needed to retrieve the file is reduced. The savings in access time on an "optimized" file averages to approximately one to six milliseconds, depending on the type of disk drive.

By placing the most frequently accessed files on a disk (the "hot" files), near the center of the disk, it is argued that the disk read head is, on average, closer to the data. In order for a software product to do this, several components must be taken into account. Here is a simplified list of the actions necessary for file optimization:

- Detecting the "hot" files
- Analyzing the present location of the "hot" files
- Deciding where to place the "hot" files
- Detecting the seldom-used ("cold") files
- Moving the "cold" files to the edges of the disk
- Moving the "hot" files to the center of the disk

All of this activity requires a variety of system resources. Detecting "hot" and "cold" files can consume large amounts of CPU resources, as can analyzing and deciding what to do with the files. Moving the files to their new locations consumes CPU and I/O resources. Typically, optimization puts such a load on the system that the system manager schedules optimization to be done at night or outside of production hours.

This introduces another series of problems with optimization. System resource usage is only *one* of the problems with "optimization".

Regardless of where files are placed, as the placed files are accessed and new data is added to the file, the added data is stored in the first available free space on the disk (as per the extent cache). The chances of the free space being contiguous to the placed file are very slim indeed.

User access to disks is usually random. Just when a system manager thinks a disk is nicely arranged, some user returns from vacation, logs in and wants to read their mail files just after the "optimizer" tucked them away because they were not used very much. Now the "optimizer" senses that these mail files are active ("hot"), and tries to move them to the center of the disk, but first, it must move one or more other (previously "hot") files *from* the center of the disk. After the user is done reading the mail files, chances are good that the files will not be read again for quite some time. Now, they are "cold" files placed near the center of the disk, which will need to be moved again to make room for the next file the "optimizer" determines is "hot". This becomes an endless cycle on an active disk.

It is likely that the data determining how "hot" a file is will change very quickly. Does the system manager set the "optimization" software to try to keep up with it? Or does he allow it to collect a day's worth of data and average the results for overnight "optimization"? If he allows the file movement to be too frequent, the system resource load imposed by the file movement alone will be extraordinarily high. On the other hand, if files are only allowed to be moved overnight, the "optimizer" will always be a day behind (Tuesday's transaction file will be moved into the middle of the disk just in time for Wednesday's workday). Here again, it is vital for the system manager to analyze the true cost of repetitious file movement.

Then there is the issue of physical block locations as opposed to logical block locations. When you force a file to a specific position on the disk by specifying exact logical block numbers (LBNs), how do you know where it *really* is? Some people would have you believe that logical blocks are the same as physical blocks, but they are not. LBNs are assigned to physical block numbers (PBNs) by the disk's controller. Disks supplied by Digital often have as many as 10% more physical blocks than logical blocks. The LBNs are assigned to most of the physical blocks and the remainder are used as spares and for maintenance purposes. Keep in mind that magnetic disks are far from perfect and blocks sometimes "go bad". In fact, it is a rarity for a magnetic disk to leave the factory without *some* bad blocks. When the disk is formatted, the bad blocks are revectored to spares. In other words, the LBN assigned to the bad block is reassigned to a different physical block. This revectoring can also be done on the fly while your disk is in use. The new physical block *may* be on the same track and physically close to the bad one, but then again, it may not. The bottom line is that LBNs do not correspond to the physical block of the same number and two consecutive LBNs may actually be widely separated on the disk, so you can never be certain exactly where your files are on a disk.

There are isolated instances where placement of one or more files in specific places on the disk can be beneficial, like certain realtime laboratory environments where a single process has continuous control of the system. In this type of system, time consumed by head movement from one particular file to another can be critical to the success of the process. The system designer can minimize that critical time lag by calculating the ideal location for the second file in relation to the first and forcing the two files to exact locations. Then, when the process has completed reading the first file, access to the second file is done with minimal delay.

In a nutshell, optimization is not all that optimum, except in a few isolated cases.

A much better solution for handling "hot" data is to use a data caching product like *I/O EXPRESS*® from Executive Software. Instead of placing "hot" *files* at some special location on a disk, *I/O EXPRESS* takes a completely different approach. *I/O EXPRESS* places frequently accessed *blocks* of data into memory on the VAX system, so that requests made later for the same blocks of data may be satisfied from memory, instead of from disk. The generic term for this activity is *data caching*. Data is retrieved from a cache much faster than from a disk. By caching data blocks in memory on the VAX system, the performance of the VAX is improved considerably.

With *I/O EXPRESS*, there is no need for the system manager to predict which files (or blocks of data) should be cached. Selection of the disk blocks to hold in memory is done completely automatically by *I/O EXPRESS*. In most cases, a large file would not be stored in the memory-based cache in its entirety; only the blocks of the file most likely to be read soon would be there. This makes more memory available for the caching of other blocks of data, as well as for other system or application needs.

Data caching offers much greater system performance improvement than file optimization. At best, an optimized file can be accessed in about 1 to 6 milliseconds faster (on average) than a file close to the edge of the disk. On a disk with an average access time of 25 milliseconds, optimization could *potentially* reduce the average access time to 19 milliseconds.

On the other hand, data can be retrieved from a cache in memory in approximately 0.5 to 2 milliseconds. That is an order of magnitude faster than an optimized file. No file optimization software can speed up a VAX system the way *I/O EXPRESS* does!

Best of all, *I/O EXPRESS* accomplishes all of this at virtually no cost in terms of system resources. The memory used by *I/O EXPRESS* would not otherwise be used, and the VAX system's capacity to handle data is increased dramatically.

9 Preventing Fragmentation

The old saying, "An ounce of prevention is worth a pound of cure", is certainly true when it comes to disk defragmentation. There are several steps you can take to slow down fragmentation, as well as a new product available that can dramatically reduce the fragmentation on your disks as files are created and extended.

9.1 System Management Techniques

Basic good system management techniques can help alleviate some of the fragmentation problem.

9.1.1 Keep Sufficient Free Space

Clearing unneeded files from the disk is one of the most effective means of preventing fragmentation. The more space you can keep free on the disks, the less likely they are to become fragmented. It should be noted, however, that reducing capacities below 50% does not improve things much – not enough to warrant the cost of maintaining so much unutilized disk space. On the other hand, if you cannot keep the disks half empty, keeping them 40% empty, or 30% or even 20% helps a lot. Generally, disk I/O performance is great when the disk is 50% empty or more. Performance worsens slightly as the disk fills to 80% of capacity. Above 80%, performance really goes down hill and, if you are running above 90% full, you have a built-in performance problem of severe proportions.

When you see a disk above 90% full, you should lose interest in getting the disk defragmented, and instead concentrate on getting some space cleared off the disk. The performance gains will be clearly visible. It is not too difficult to free up 10% of a disk. One effective method for doing this is to issue a notice to all the users (via a NOTICE file in the login procedure). The notice should say something similar to "The disks on our system are too full. Delete all unnecessary files from the system within 24 hours. If sufficient space is not freed up by then, the system manager will delete files from the directories of the worst offenders until sufficient space is available. Files not needed now which might be needed later can be stored in archives until they are needed."

A notice like this will often get as much as 20% of the disk freed up. Of course, you may have to actually enforce it from time to time, but a check of each user's files usually reveals one or two users who have dozens and dozens of versions of old, old files that they have not touched in months. (This is the guy who saves *everything* and consumes as much space as half the other users combined.)

9.1.2 Specify large enough directory size

The DCL INITIALIZE command allows you to preallocate space for directories. Unfortunately, it defaults to only 16 directories, so most disks require additional directory file space to be allocated. The additional directories are created right in the middle of production processing, disrupting application I/O and scattering the newly-created directory files all over the disk.

To get around this problem, when you initialize a disk, estimate how many directories will be created on it, and specify a slightly larger number with the INITIALIZE /DIRECTORIES=n qualifier.

9.1.3 Specify Enough Space for INDEXF.SYS

Another tip is to specify enough space for the INDEXF.SYS file when you initialize the disk.

The DCL INITIALIZE command allows you to preallocate space for file headers in INDEXF.SYS. Unfortunately, it also defaults to only 16 files, so most disks require additional space to be allocated to INDEXF.SYS after the disk has been initialized. The extra space allocated is often not contiguous to INDEXF.SYS, so this all-important file becomes fragmented and performance suffers. Since this file is always held open, defragmentation software has no effect on this file.

When you initialize a disk, estimate how many files will be created on it and specify a slightly larger number with the INITIALIZE /HEADERS= qualifier.

9.2 Preventative Software

Defragmentation software corrects fragmentation after the fact; it does its job after the fragmentation has occurred. Since November of last year, there has been a new product, unique in the market, which was designed to *prevent* fragmentation (or at least a good percentage of it) *before* it occurs. This product is named Fragmentation Controller™.

Fragmentation Controller intercepts file open, creation and extension requests. The parameters given in each request are examined. If the OpenVMS system default values are given, and Fragmentation Controller determines that the default values would cause the file to be

fragmented, it modifies the values. The values are modified by requesting *contiguous best try* file allocation, and by requesting an optimum file extension size.

Fragmentation Controller was developed by Touch Technologies, Inc., and is being sold exclusively by Executive Software.

One of the first questions that comes up is, "Do I need a defragmenter if I'm using Fragmentation Controller?" The answer is "Yes." Fragmentation Controller needs contiguous free space in order to perform its job effectively. As files are deleted, free space becomes fragmented, even when file creations and extensions are done contiguously. To create the contiguous free space needed, it is still necessary to run a defragmenter. But, since the disk is less fragmented overall, the defragmenter does its job using fewer system resources, and should do it in less time (or less frequently).

Another common question is, "Why do I need this type of product when I'm already using a defragmenter?" Keep in mind that a defragmenter does its work after the fact — after the fragmentation has occurred. When it is done, fewer I/O are necessary to *retrieve* files from the disk. Additionally, since free space is contiguous, file creations are likely to be contiguous — for a while. But there is a catch — *as soon* as you start deleting (or purging) files from the disk, the extent cache starts getting loaded with locations of all this "new" free space. Regardless of the amount of contiguous free space you have, the OpenVMS file system will start filling the spaces left by the recently deleted files *even if it means storing the file in a fragmented condition.* Fragmentation can start as soon as you delete files. By intercepting the file creation and extension requests, Fragmentation Controller can evaluate the variables involved and, if possible, cause the file to be created or extended contiguously.

Another point to be aware of is this: As you know, disk I/O is increased when reading a fragmented file, thus slowing down not only that process, but potentially other process as well (remember the I/O request queue). Not surprisingly, the *same* additional, unnecessary I/O traffic is caused when writing a fragmented file as well. To trim it down to basics — A defragmenter saves I/O traffic on reads, Fragmentation Controller saves I/O traffic on writes. This is not just conjecture, it is measurable in elapsed times. Cutting down on the unnecessary I/O traffic can and will speed up your system.

This product is a great companion to a disk defragmenter such as DISKEEPER/*Plus*. It reduces fragmentation on files that a defragmenter does not usually process. Files like open files, which a defragmenter would back off from, can be extended contiguously. For example, virtual disks used by Pathworks™ users are treated as open files while mounted and therefore are not defragmented. Fragmentation Controller keeps them contiguous if at all possible. Also, defragmenters typically do not address work files, print files, mail files, and other temporary files that are created in a fragmented condition, used fragmented, and then are deleted before the defragmenter runs. These types of files will no longer cause unnecessary additional I/O when Fragmentation Controller is running.

10 In Conclusion . . .

This report has covered many of the issues surrounding disk fragmentation. It has shown that fragmentation is unavoidable unless certain measures are taken *before* the fragmentation starts. We have also examined some ways to see how much fragmentation you have, and looked at a variety of ways to lessen or eliminate it.

To sum up:

- If you are using a VAX or an Alpha AXP, you have fragmentation
- Backup and restore is *not* a viable solution to fragmentation
- Buy a defragmenter — you need it
- Data caching will provide much more benefit than "file optimization"
- A fragmentation preventer will enhance your defragmenter and provide additional reduction of unnecessary I/O overhead